Advanced Topics in Machine Learning: Convolutional Networks (Part 2)

Aaron Adcock and Laurens van der Maaten

Instructors

Aaron Adcock

Laurens van der Maaten

… and all your favorite TAs!

Summary from last time

- Loss measures discrepancy between a prediction and the true label
- ERM minimizes the loss averaged over the training data
- Logistic regression is an linear ERM model that uses logistic loss
- Multi-layer networks iteratively apply learned linear functions and fixed non-linear functions: gradients computed by chain rule (backprop)
- Convolutions filter signals and are the basis of convolutional networks

Multilayer networks

- Suppose we wanted to train a multilayer network to recognize images

- What would go wrong?

Suppose the image has 256x256 RGB pixels,
and we have 1,000 classes.
How many parameters would a single layer have?

3 x 256 x 256 x 1,000 = 196,608,000

From last time

Convolutional networks

- You can think of each column in a linear layer as a "filter" for the image
- Do different parts of an image have to be filtered differently? **No!**

From last time

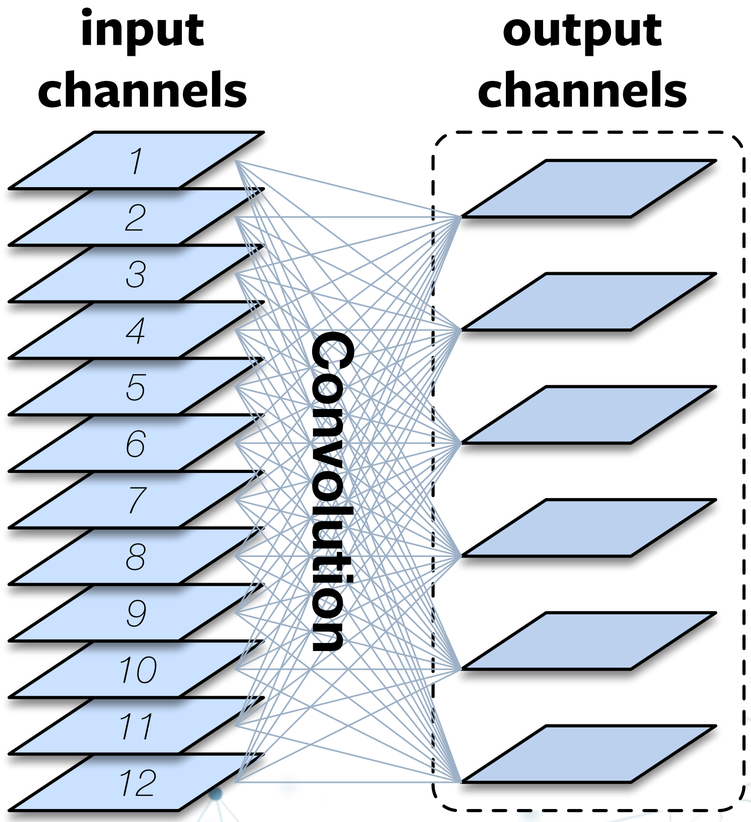
Convolutional networks

- Convolutional network layers generally have **multiple input channels** (for example, RGB) and **multiple output channels**

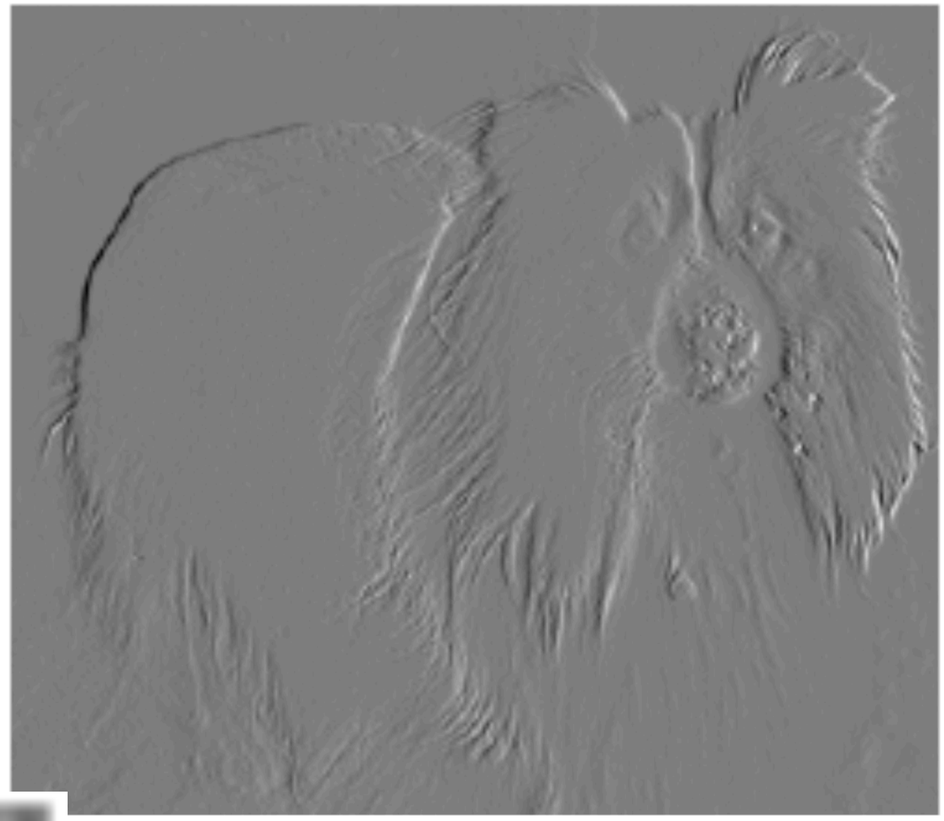
Convolutional networks

- Convolutional network layers generally have **multiple input channels** (for example, RGB) and **multiple output channels**

- To produce a single output channel:
  - **Convolve** each input channel with some **kernel**
  - Note that the kernel is **different** for each input channel
  - **Sum** the convolved input channels to produce the output channel



A simple convolutional network layer



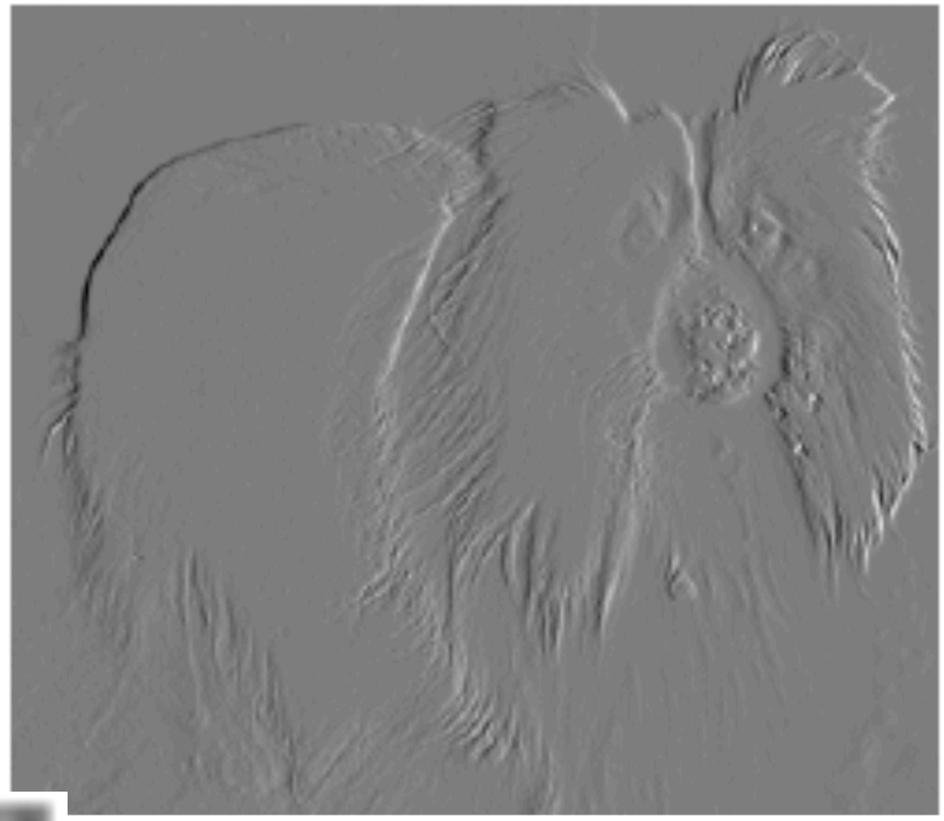
* Credits: Ian Goodfellow

A simple convolutional network layer

Only has 1 input and 1 output!

Input

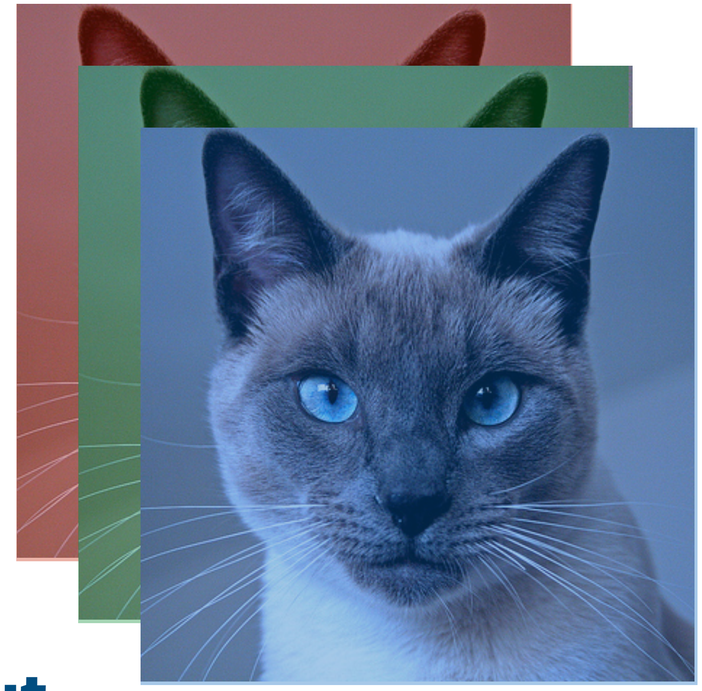
Kernel



Output

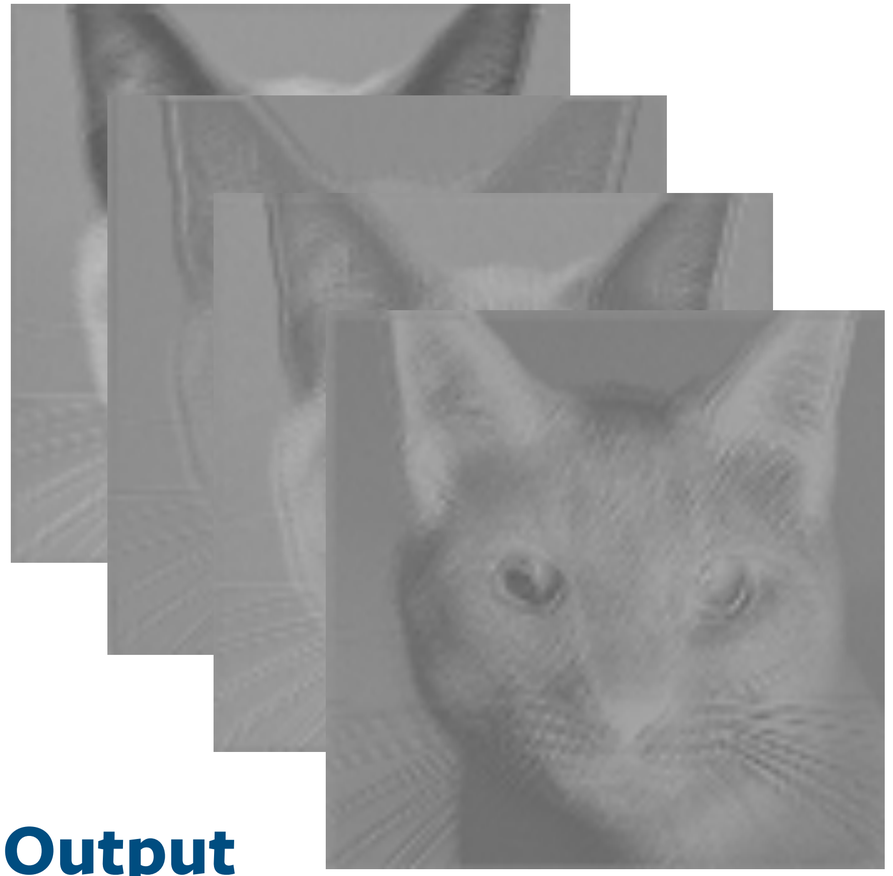
* Credits: Ian Goodfellow

A more complicated convolutional layer



Input

filter

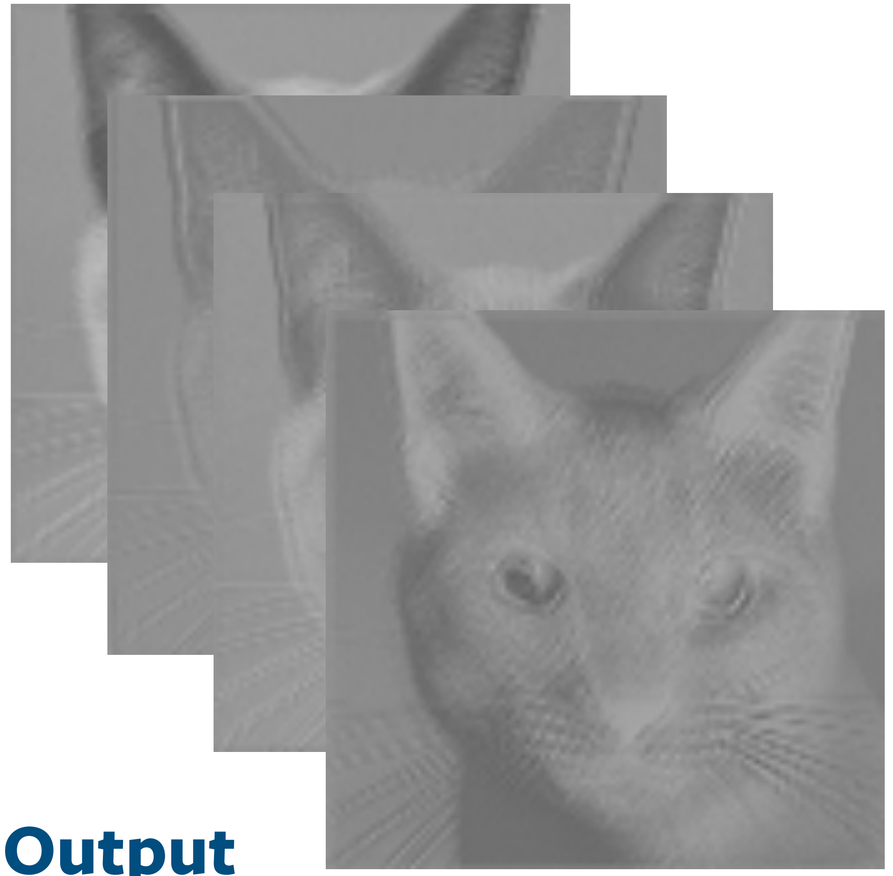


Output

A more complicated convolutional layer

Input

filter



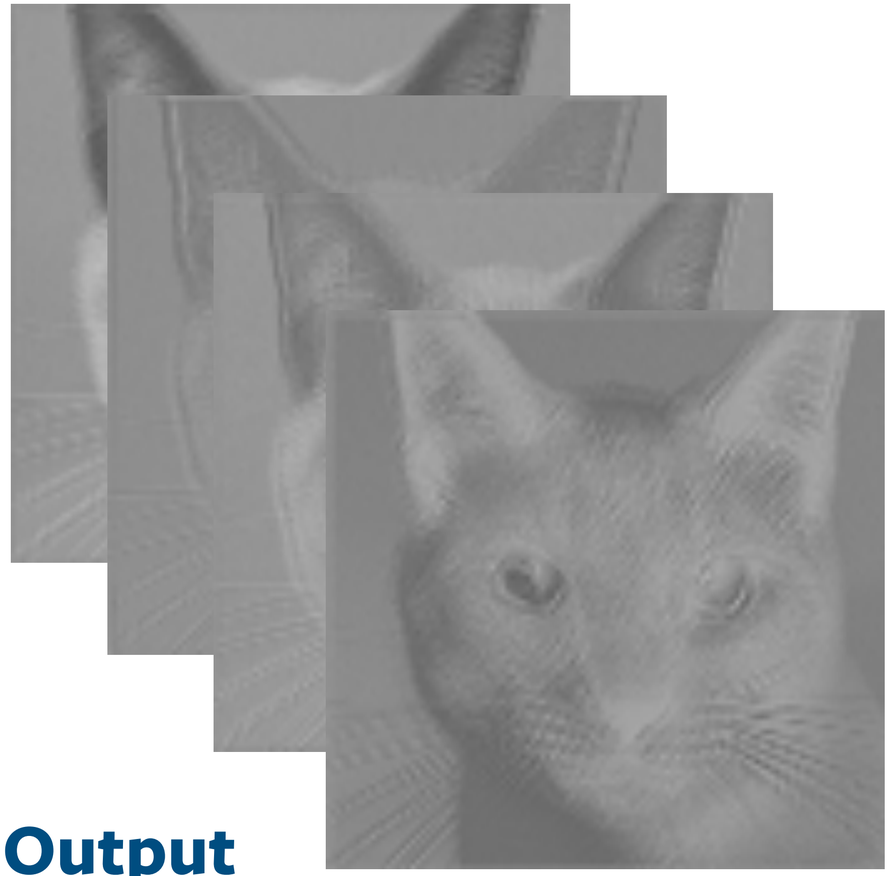
Output

How many parameters does this convolutional layer have?

A more complicated convolutional layer

Input

filter



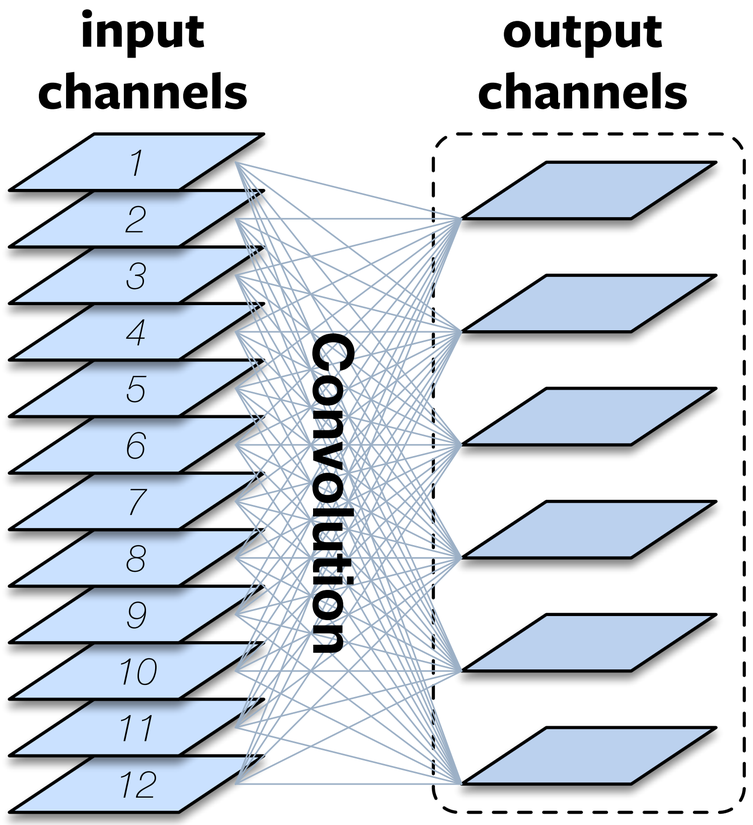
Output

How many parameters does this convolutional layer have?

3 x 4 x kernel height x kernel width

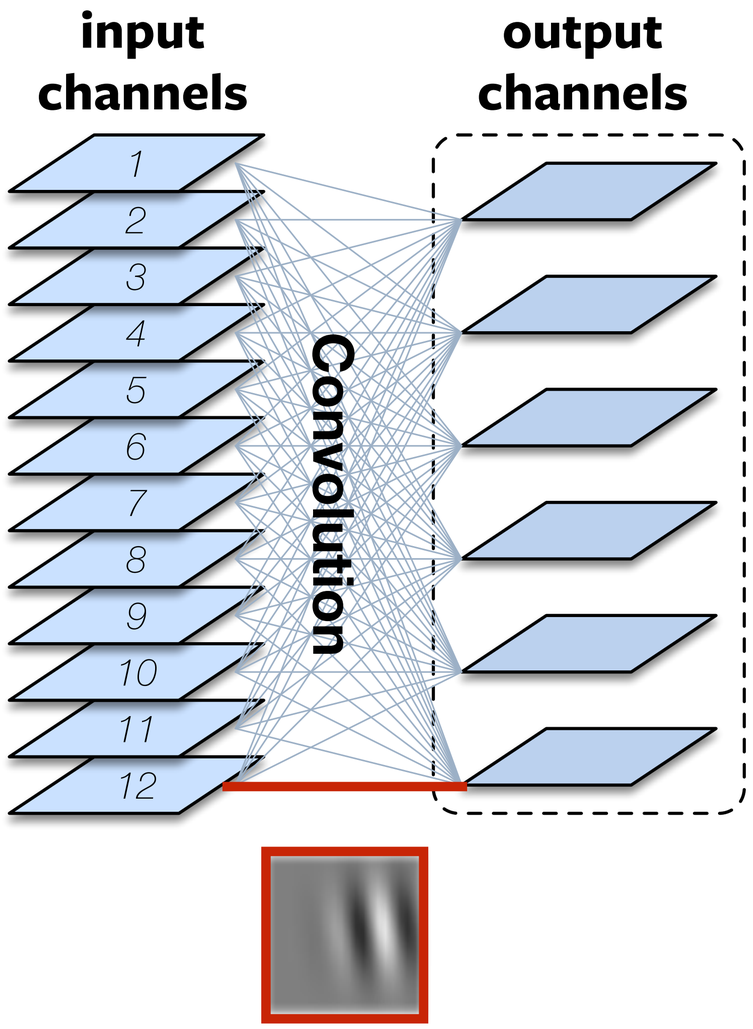
More details on convolutional layers

- Convolutional layers have five main hyperparameters:
  - Number of input channels
  - Number of output channels
  - Kernel size
  - Kernel stride
  - Input padding



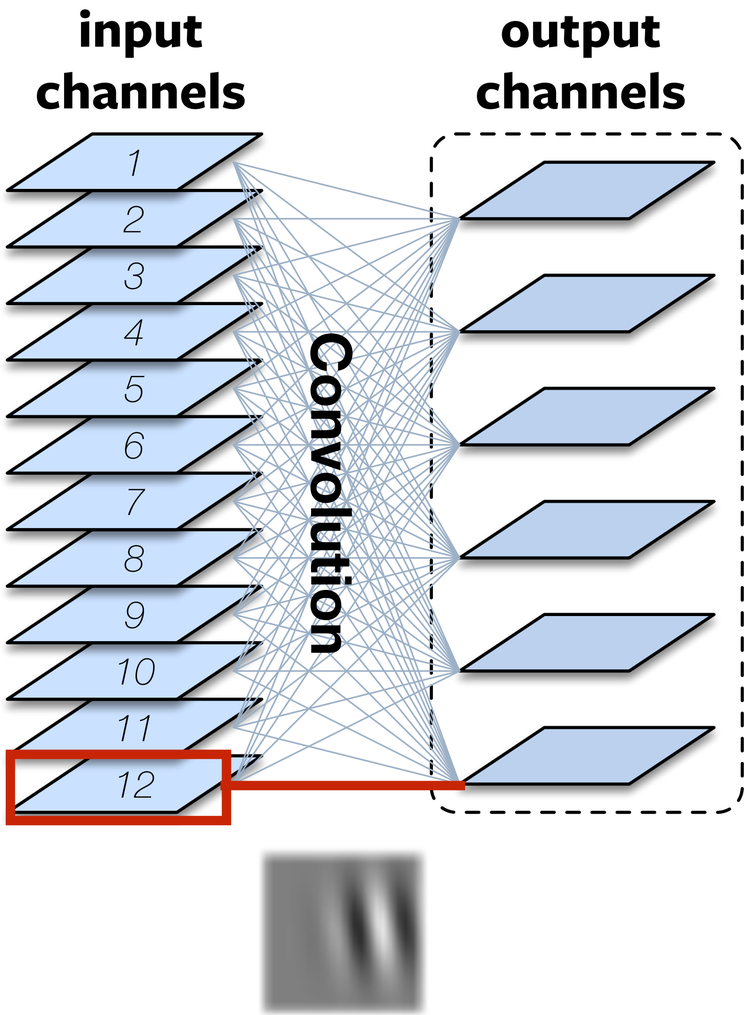
More details on convolutional layers

- Convolutional layers have five main hyperparameters:
  - Number of input channels
  - Number of output channels
  - Kernel size
  - Kernel stride
  - Input padding



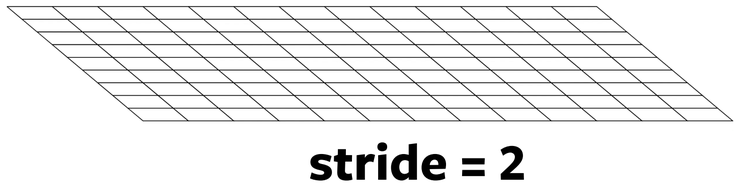
More details on convolutional layers

- Convolutional layers have five main hyperparameters:
  - Number of input channels
  - Number of output channels
  - Kernel size
  - Kernel stride
  - Input padding

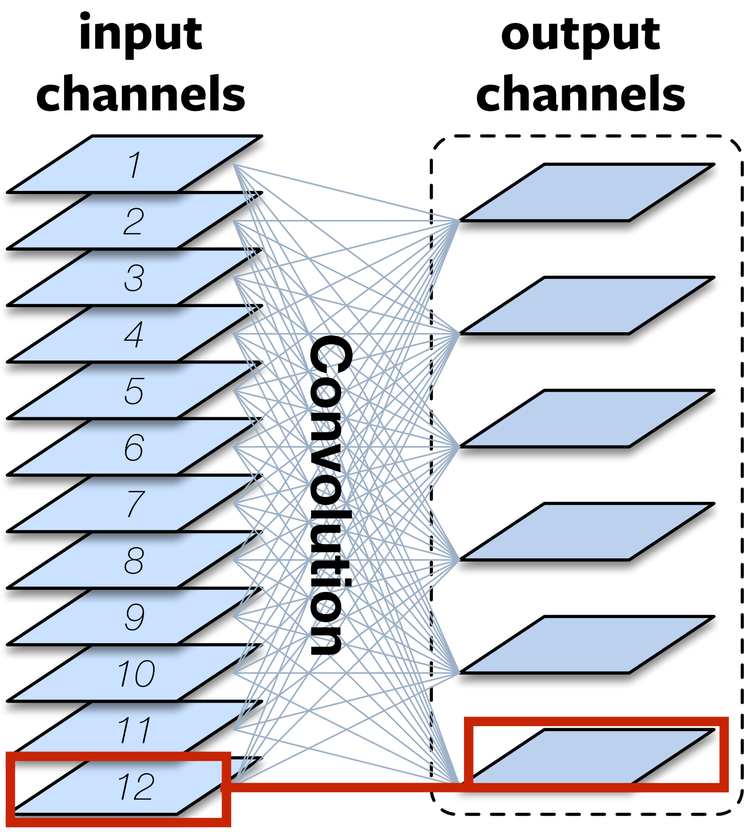


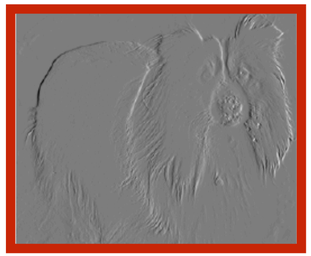
More details on convolutional layers

- Convolutional layers have five main hyperparameters:
  - Number of input channels
  - Number of output channels
  - Kernel size
  - Kernel stride
  - Input padding



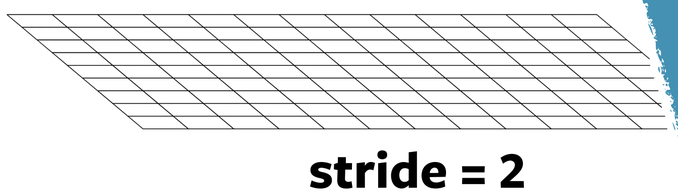
stride = 2





More details on convolutional layers

- Convolutional layers have five main hyperparameters:
  - Number of input channels
  - Number of output channels
  - Kernel size
  - Kernel stride
  - Input padding



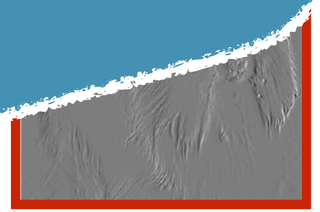
stride = 2

input channels

1

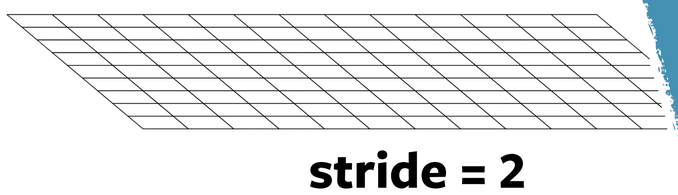
output ch

Why are we doing this?



More details on convolutional layers

- Convolutional layers have five main hyperparameters:
  - Number of input channels
  - Number of output channels
  - Kernel size
  - Kernel stride
  - Input padding



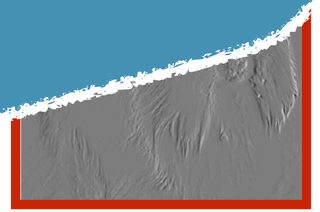
stride = 2

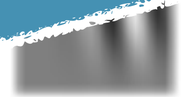
input channels

1

output ch

Why are we doing this?
Parameter-efficiency!





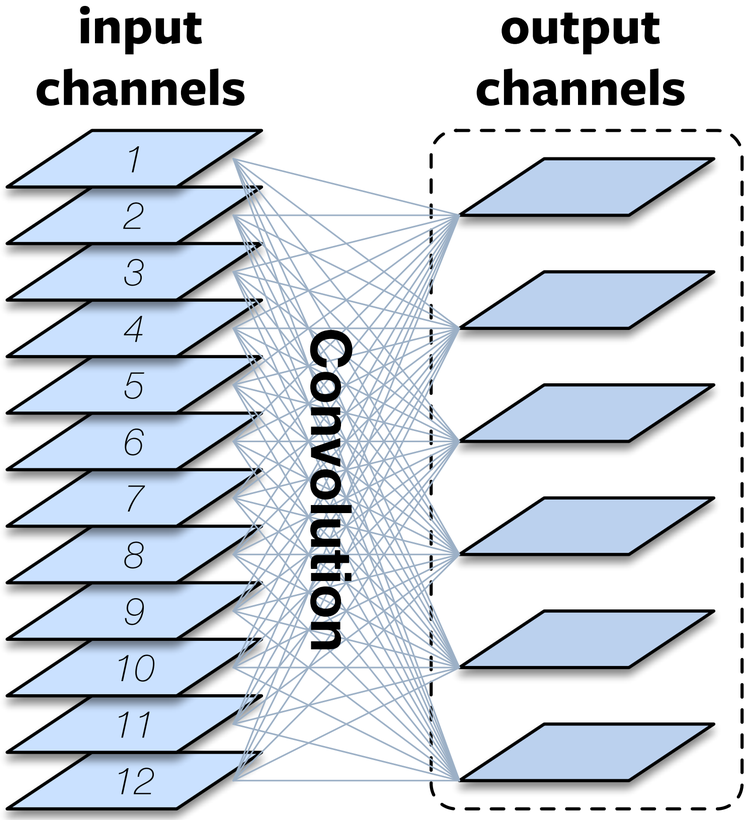
How efficient?

Let's use an example convNet:

input_size = 12, output_size = 6,
kernel size = 3x3 , stride = 1, padding = 1,
image size = 256x256

Question 1: How many params?

Question 2: How many multiplications?

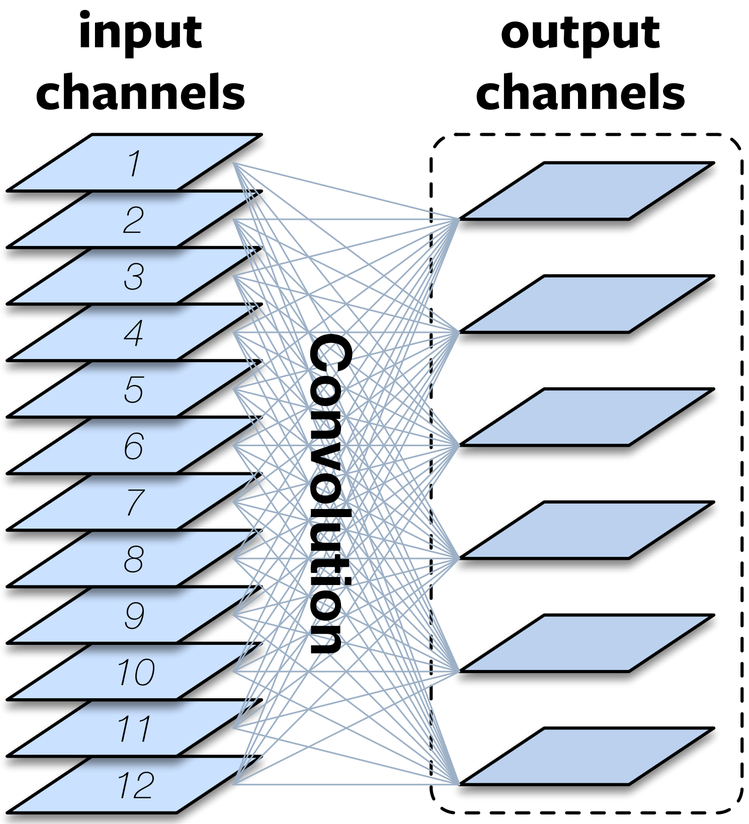


How efficient?

Let's use an example convNet:

input_size = 12, output_size = 6,
kernel size = 3x3 , stride = 1, padding = 1,
image size = 256x256

Question 1: How many params?
12 * 6 * 3 * 3 = 648 params
Question 2: How many multiplications?
648 * 256 * 256 = 42,467,328 multiplies



Is that all we need to build a modern conv net?

Is that all we need to build a modern conv net?

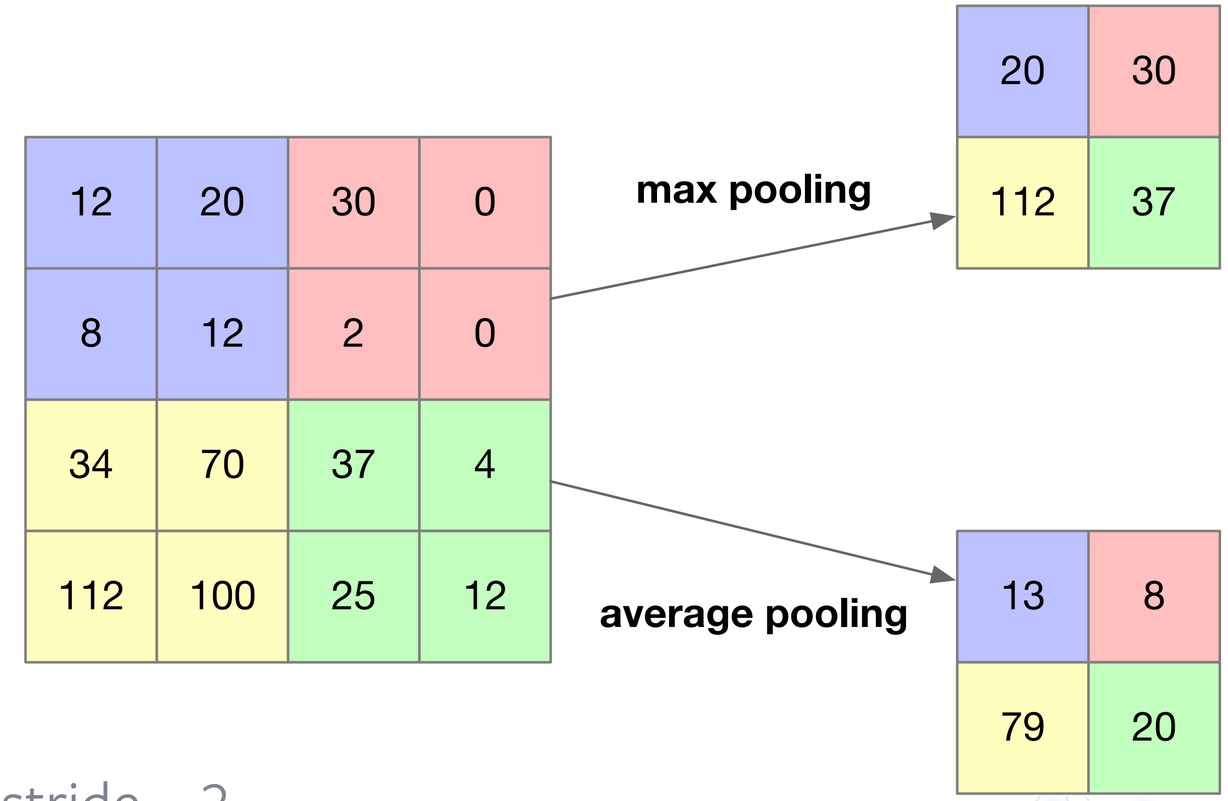
No! We also need:

- **Pooling layers:** These layers allow us to remove spatial structure to produce pooled (summarized) features
- **Rectified Linear Units:** Common non-linearity for convolutional networks
- **Batch Normalization:** Makes training more stable and we can train networks more quickly / reliably

Pooling

- Input and output of a convolution layer both have **spatial structure**

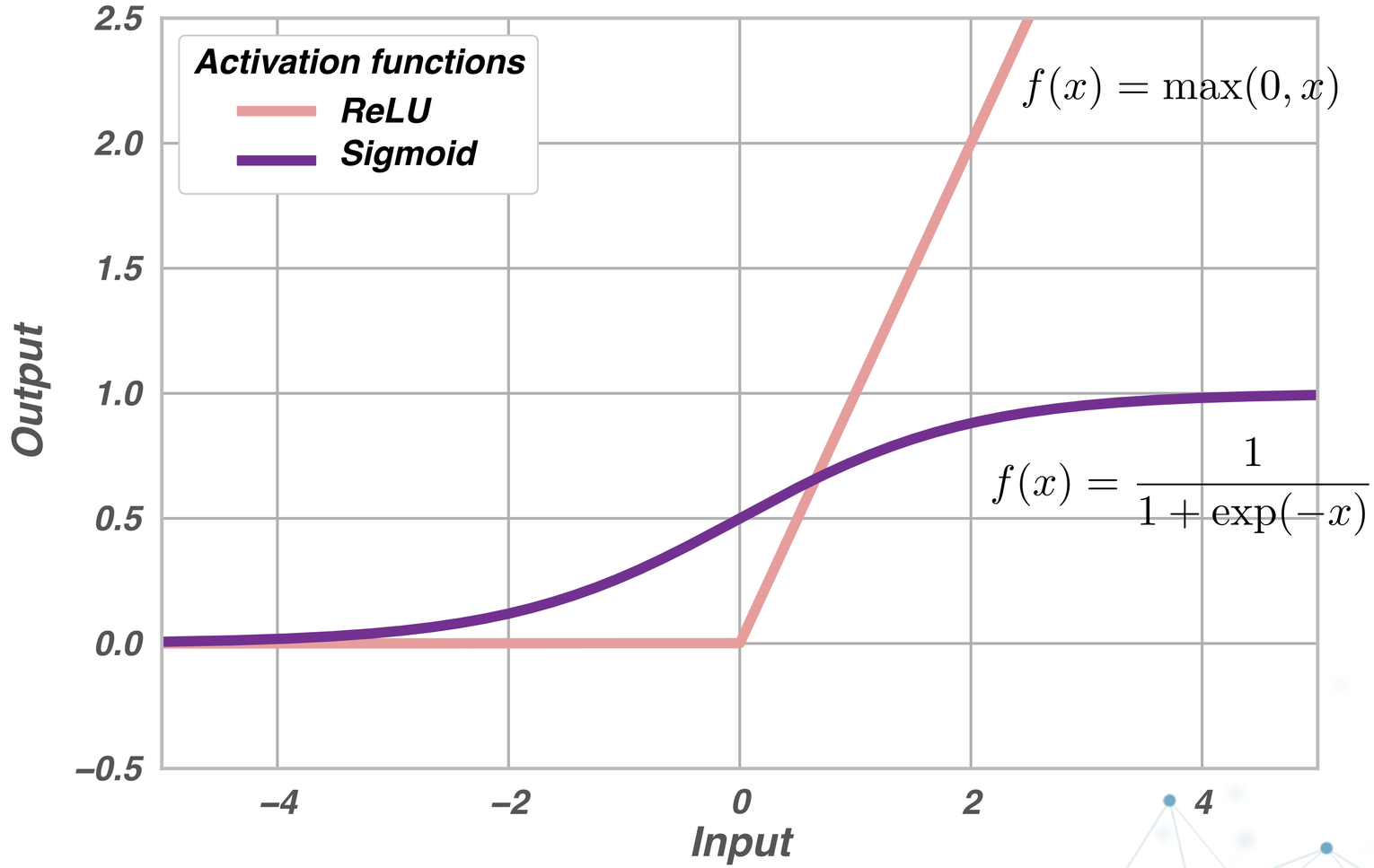
- We gradually remove the spatial structure via **pooling**:



* This pool uses size = 2, stride = 2.

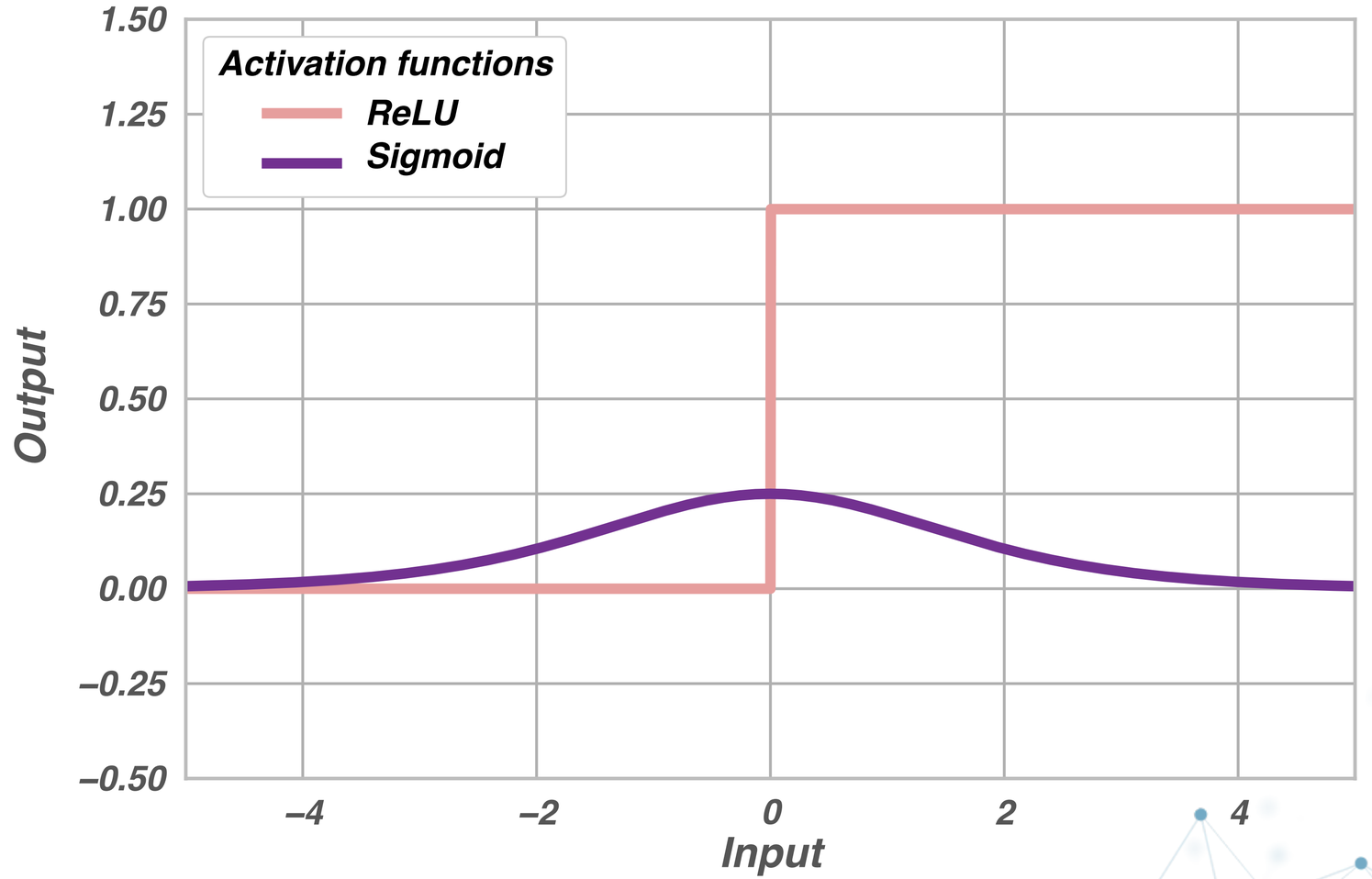
Rectified linear units

- **Rectified linear units** (ReLUs) have "better" gradients than sigmoids:



Rectified linear units

- **Rectified linear units** (ReLUs) have "better" gradients than sigmoids:

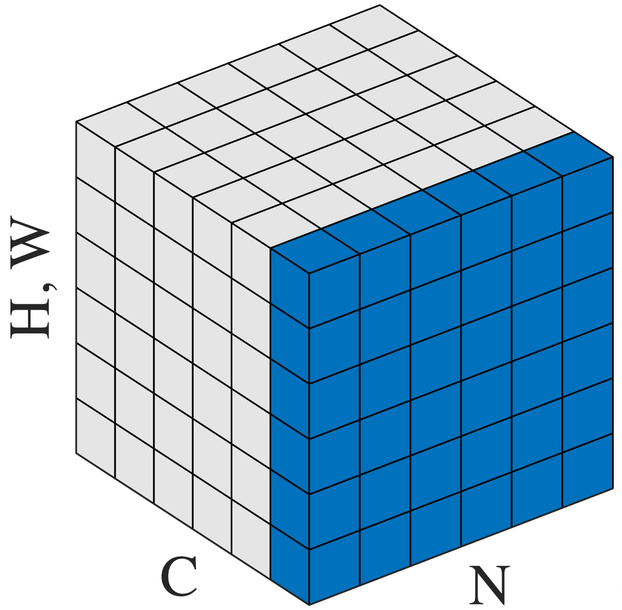


Batch normalization

- When using ReLU, gradients may easily blow up or go to zero
- **Batch normalization** make gradients better behaved:

Z-score input: $\hat{\mathbf{x}} = \frac{\mathbf{x} - \mu}{\sigma}$

Affine transform: $\mathbf{y} = \gamma\hat{\mathbf{x}} + \beta$



* Note: Batch normalization works differently in training and test time.

Batch normalization

- Makes network learning more stable and produces better final models:

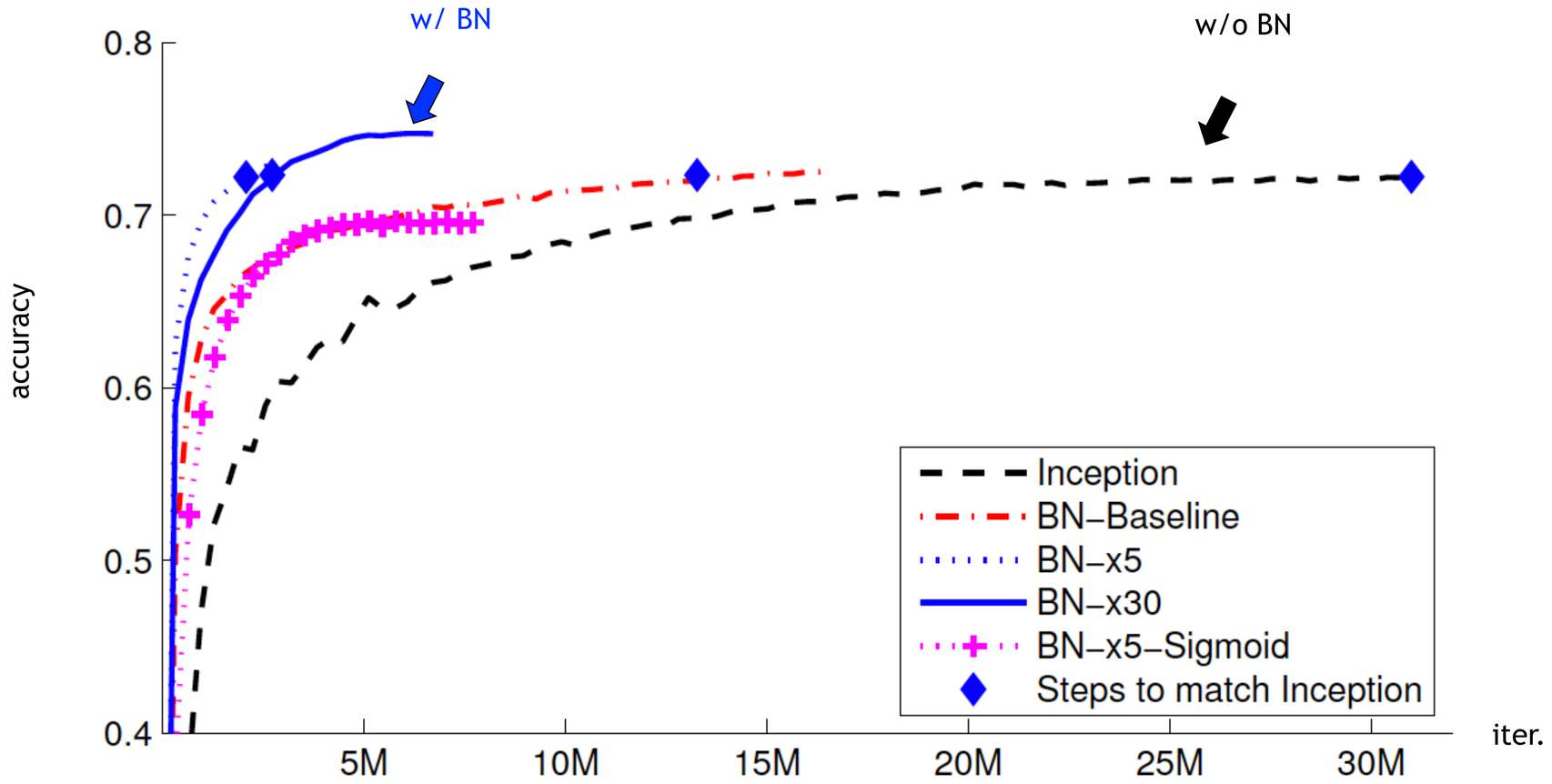
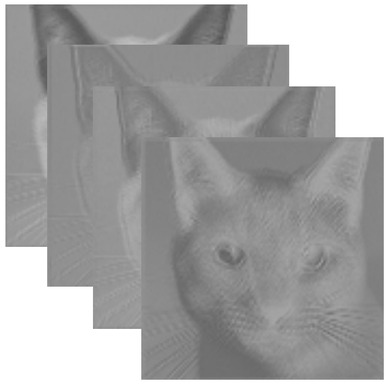


Figure credit: Ioffe & Szegedy

Convolutional networks

filter



BN +non-linearity
filter

pooling

linear
regressor

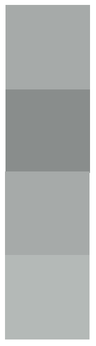
dog: **0.2**
cat: **0.7**
tree: **0.1**

compute
error

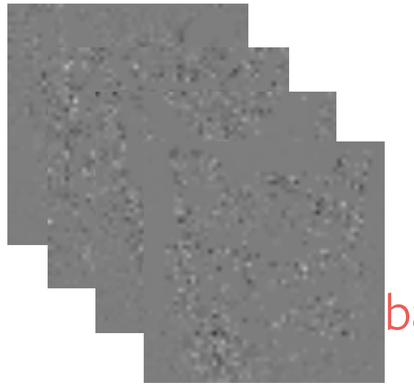
dog: **+0.2**
cat: **-0.3**
tree: **+0.1**

adapt
weights

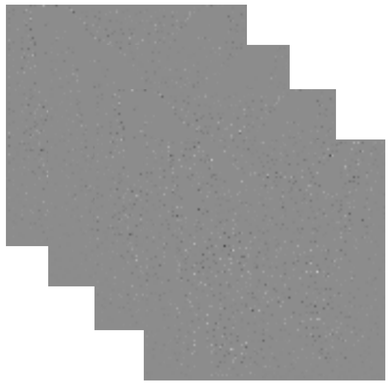
error
signal



backprop



adapt
filters



backprop

adapt
filters

error
signal

Convolutional networks

filter

adapt filters

error signal

backprop

backprop

linear regressor

adapt weights

error signal

compute error

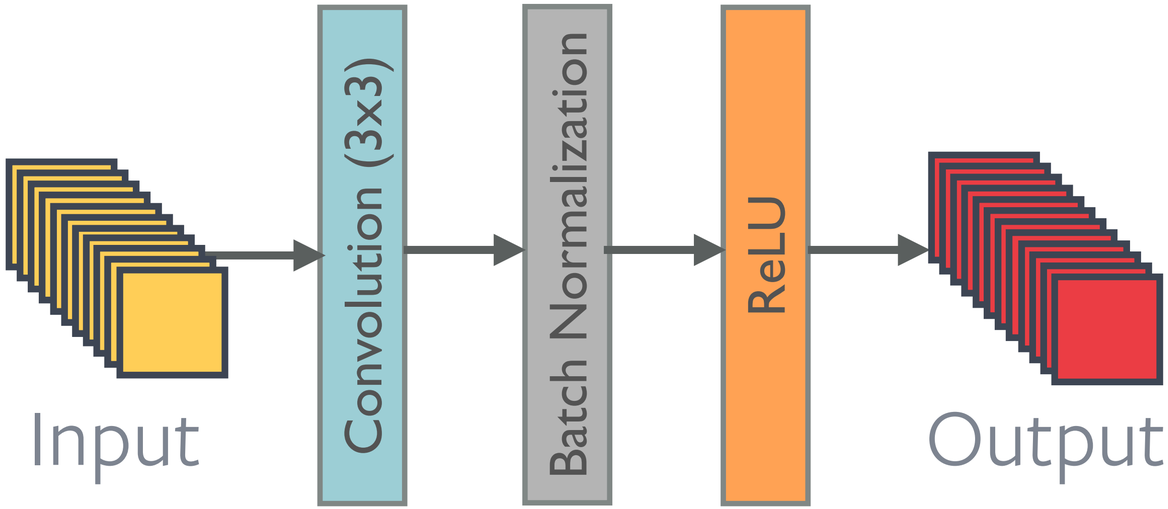
dog: **0.2**
cat: **0.7**
tree: **0.1**

dog: **+0.2**
cat: **-0.3**
tree: **+0.1**

Repeat a gazillion times for many different images

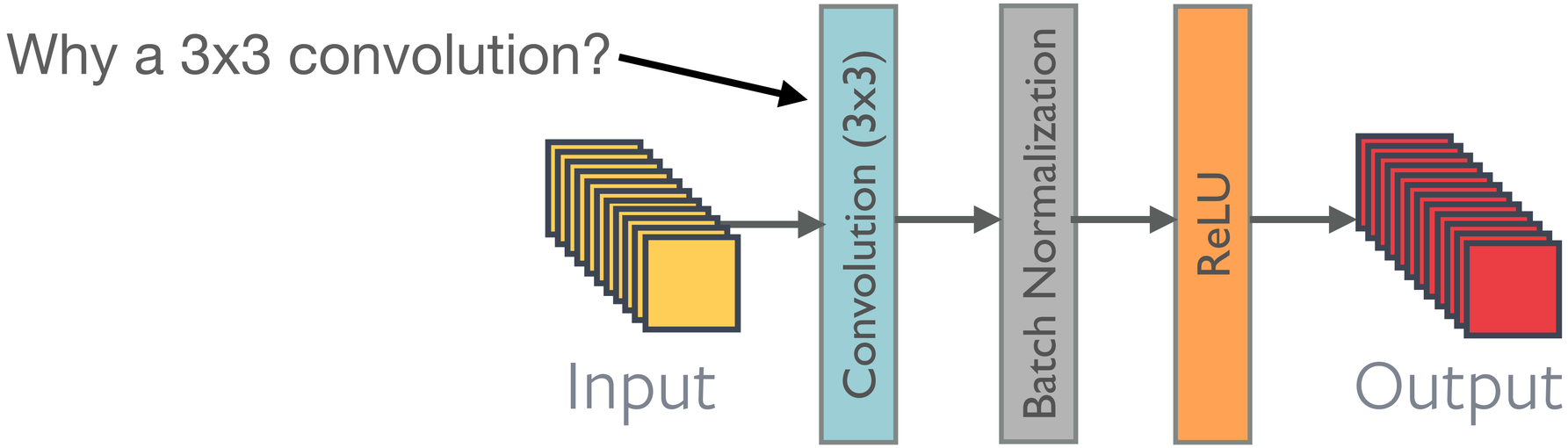
Building block

- Common building block comprises Conv2D + BatchNorm + ReLU:



Building block

- Common building block comprises Conv2D + BatchNorm + ReLU:



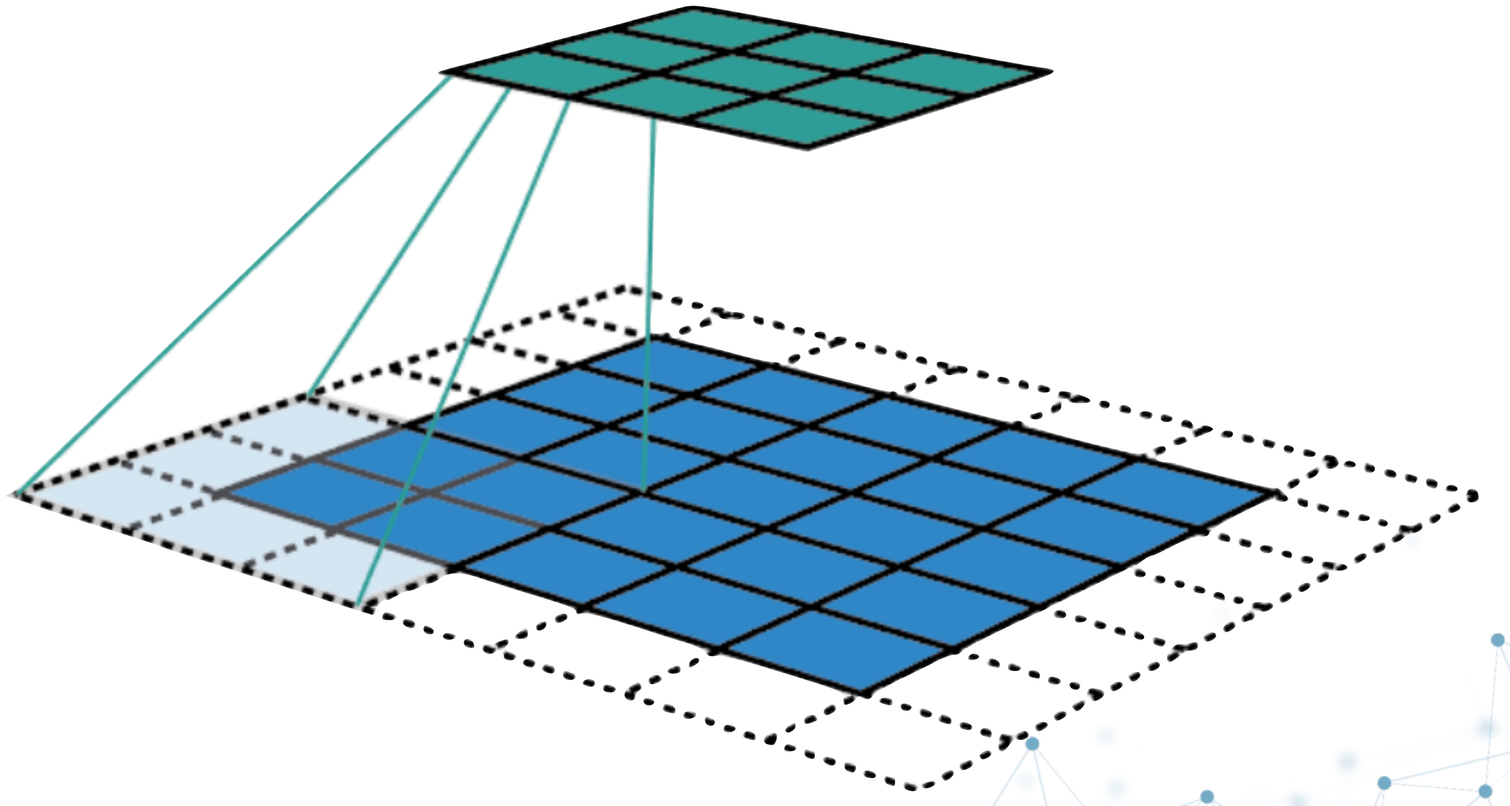
Filter sizes

- The **receptive field** of a feature is the part of the input that "influenced" the feature

Filter sizes

- The **receptive field** of a feature is the part of the input that "influenced" the feature:

Feature map

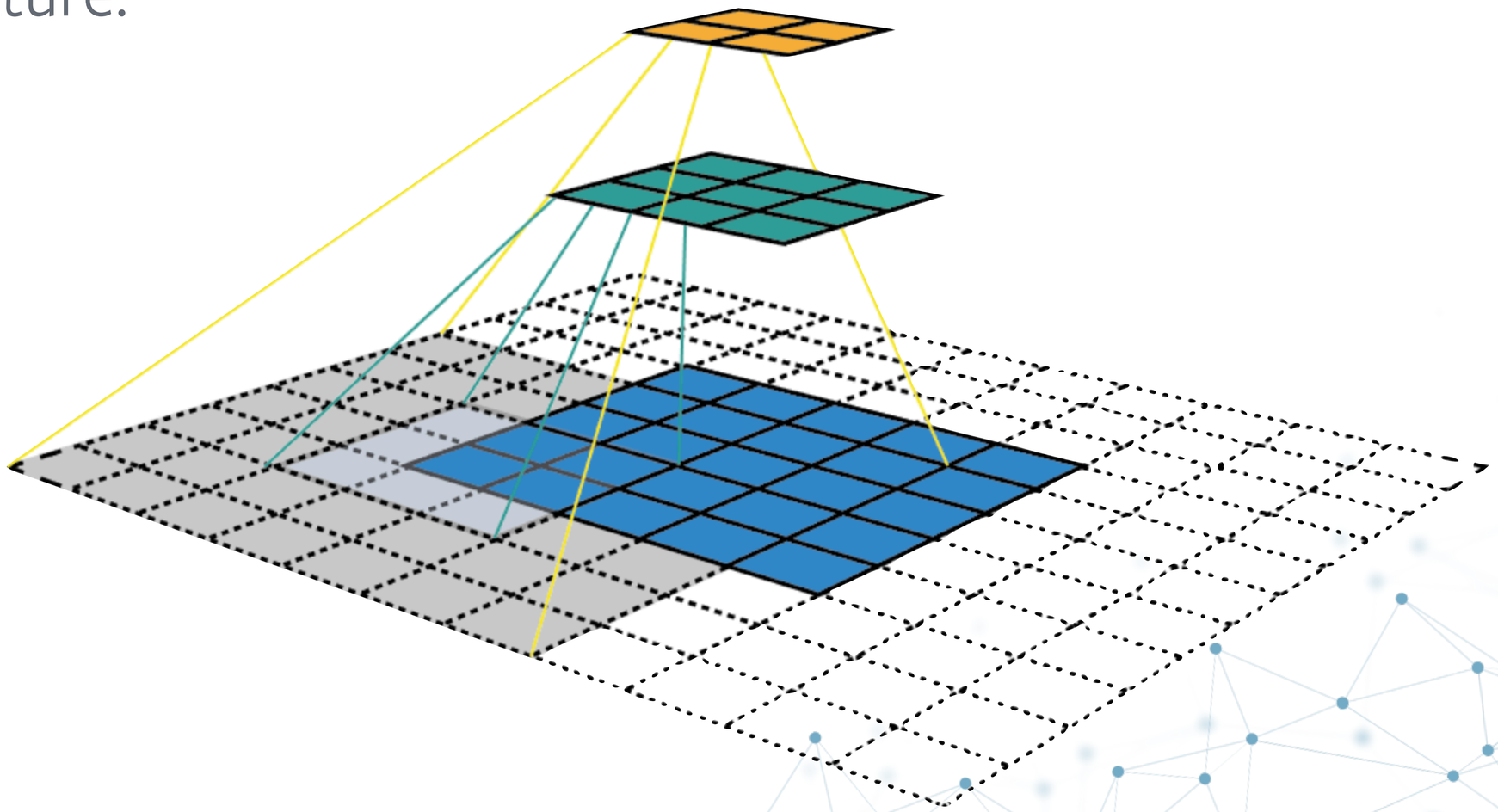


Input image

* Credits: Dang Ha The Hien

Filter sizes

- The **receptive field** of a feature is the part of the input that "influenced" the feature:



* Credits: Dang Ha The Hien

Filter sizes

- What is the receptive field of one **5x5 filter**?
  - And how much compute does it take?

Filter sizes

- What is the receptive field of one **5x5 filter**?
  - And how much compute does it take?

- What is the receptive field of **two 3x3 filters**?
  - How much compute does that take?

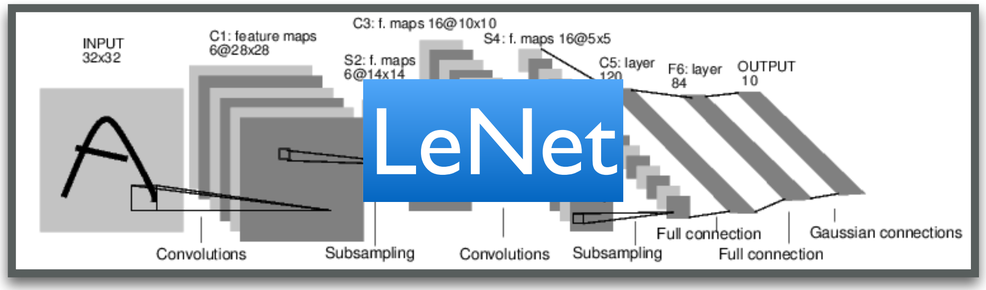
Filter sizes

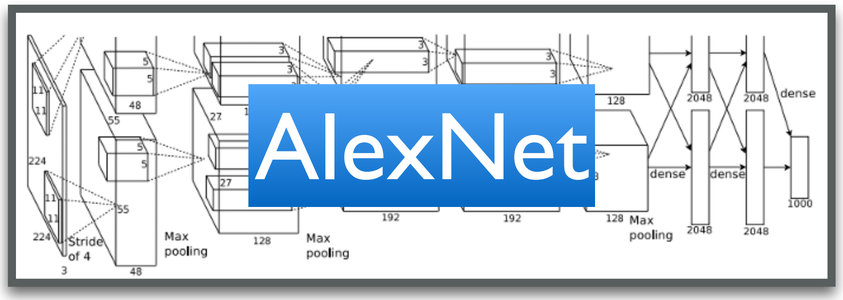
- What is the receptive field of one **5x5 filter**?
  - And how much compute does it take?

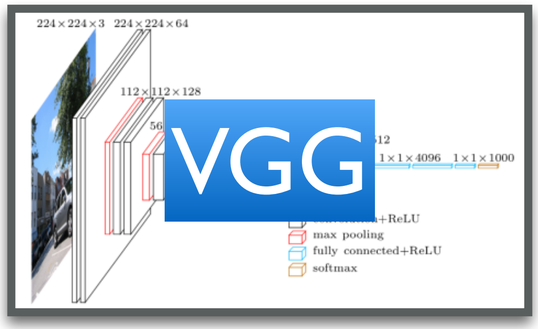
- What is the receptive field of **two 3x3 filters**?
  - How much compute does that take?

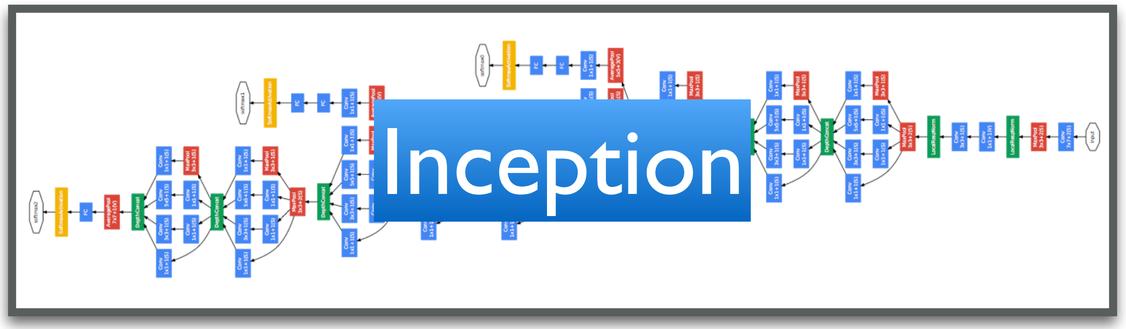
- Many current architecture use primarily **3x3 filters** for this reason

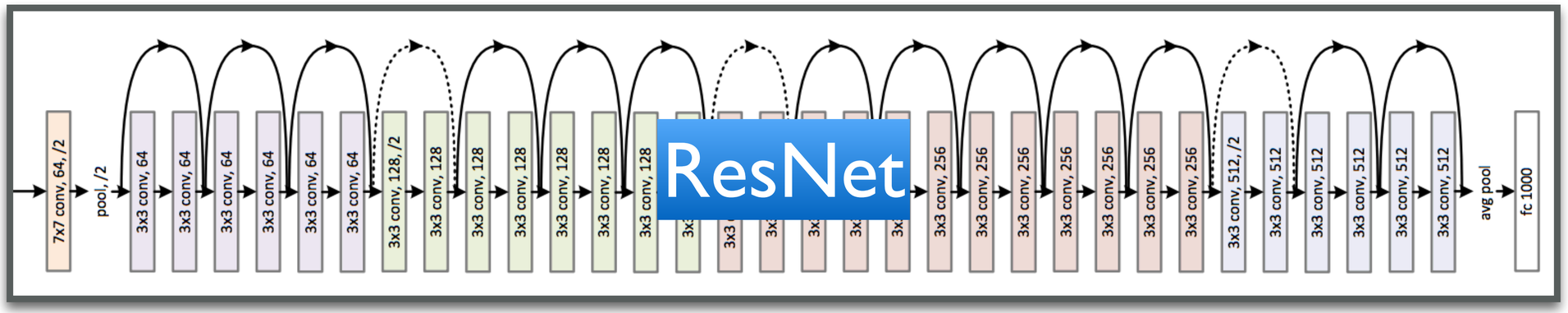
Convolutional networks







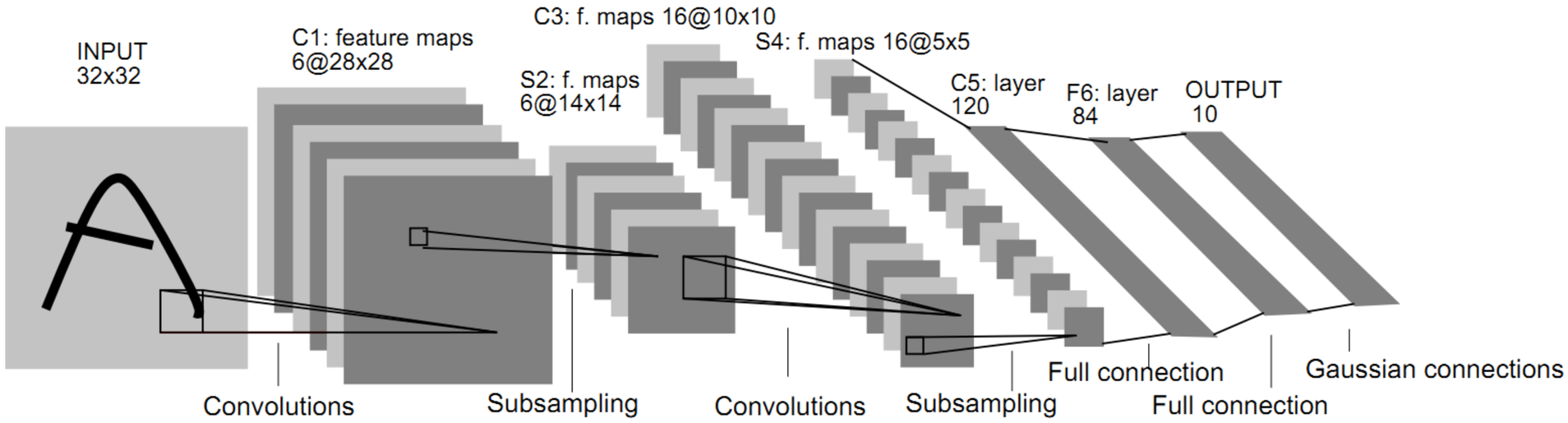




* Slide credits: Gao Huang

LeNet

- First model that really worked by using convolutions:

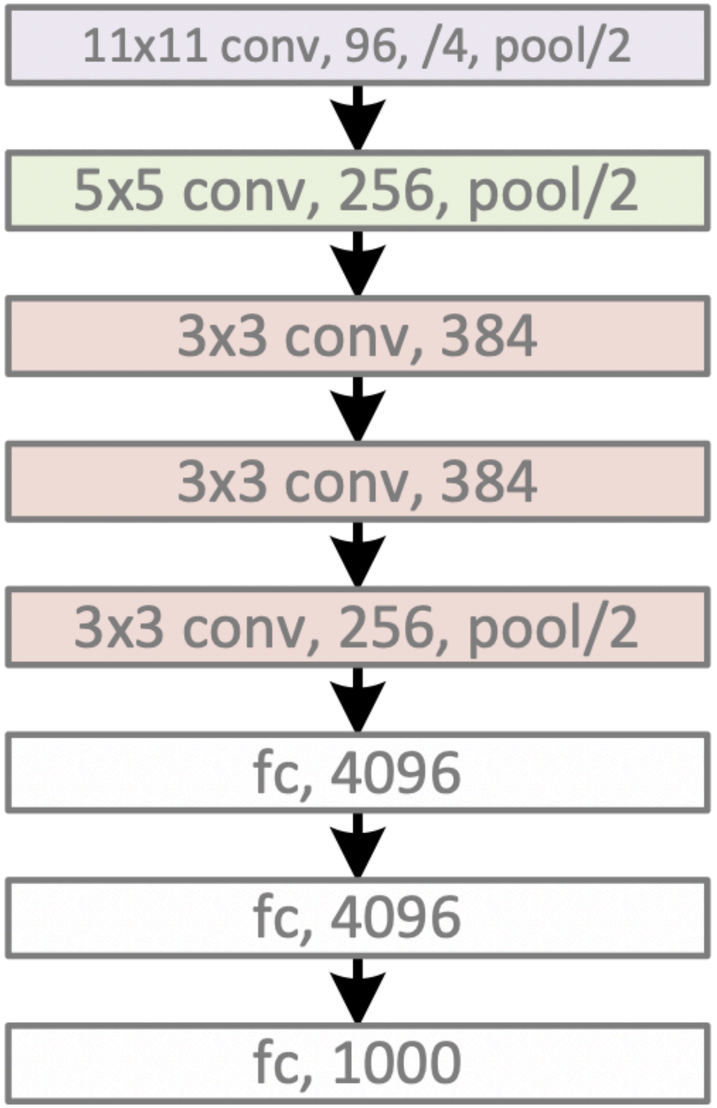


- Compared to today's networks, few convolutional layers and more fully-connected layers

AlexNet

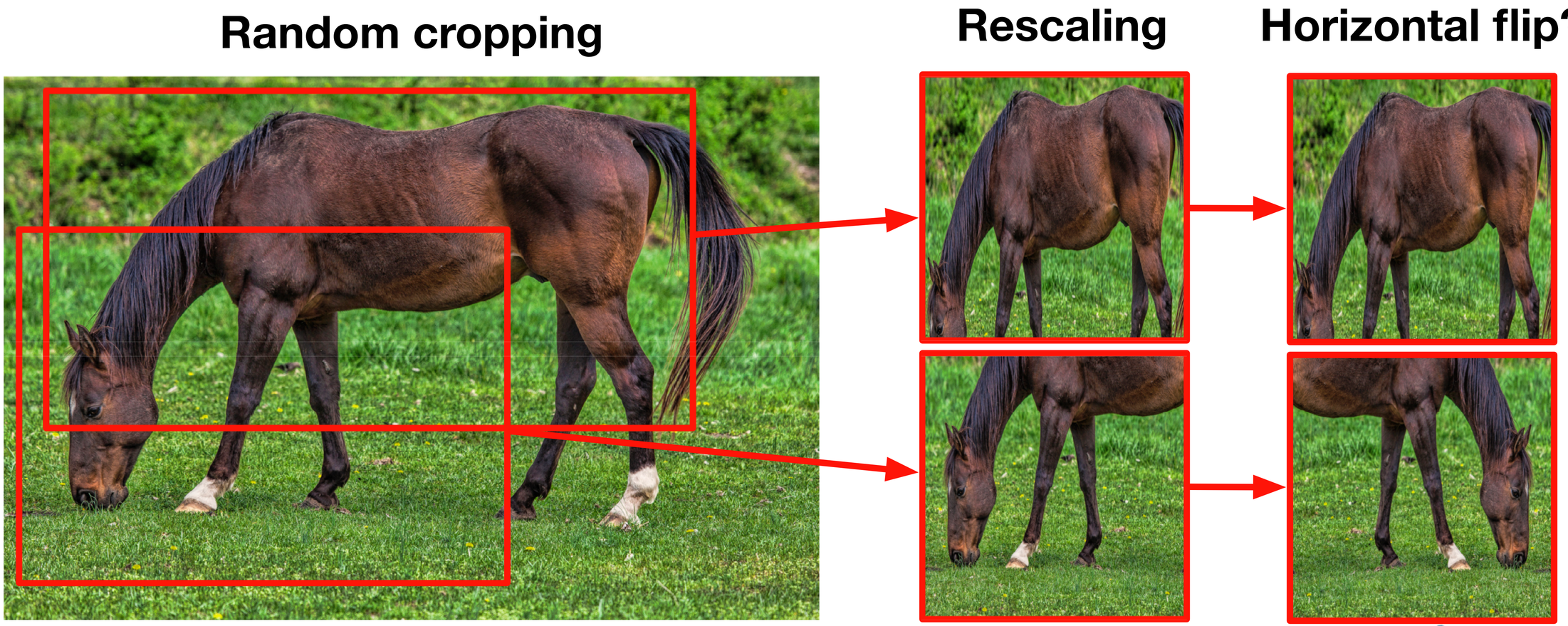
- LeNet-style model with three new components:
  - **Rectified linear units**
  - **Dropout** to reduce overfitting (currently less popular)
  - **Data augmentation** (still very important)

- AlexNet played a key role in popularizing convolutional networks



Data augmentation

- Create many (hard) training examples from a single annotated image:

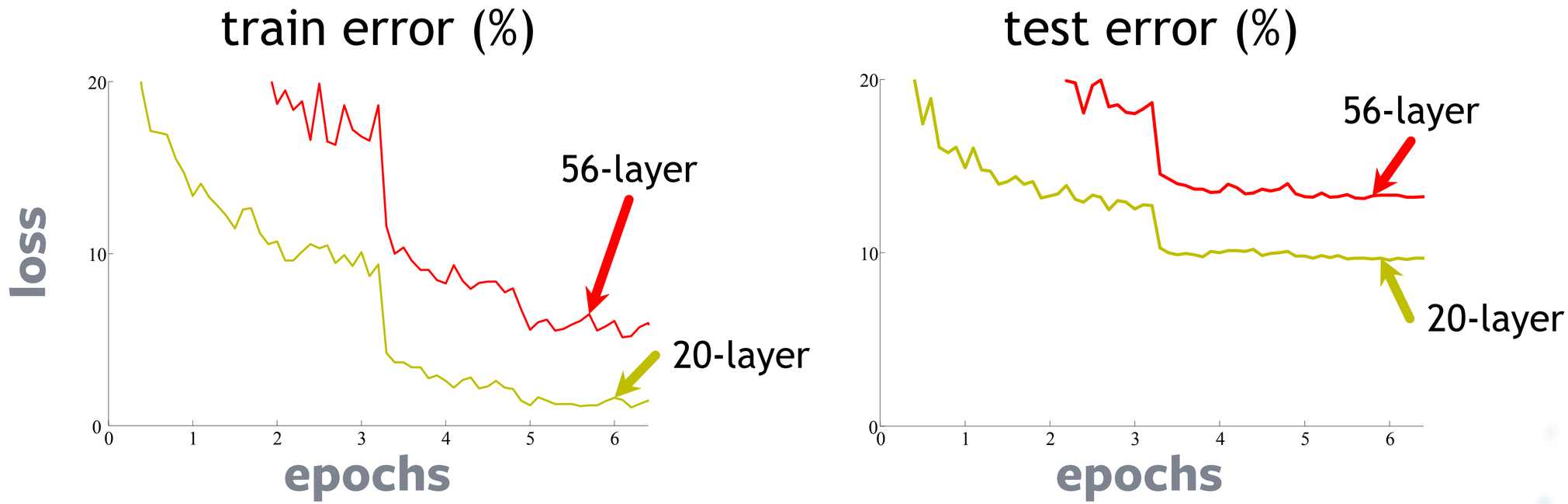


* Note: This is only used at training time! (Why?)


I WAS WINNING IMAGENET
UNTIL A DEEPER MODEL CAME ALONG
imgflip.com

Stacking layers

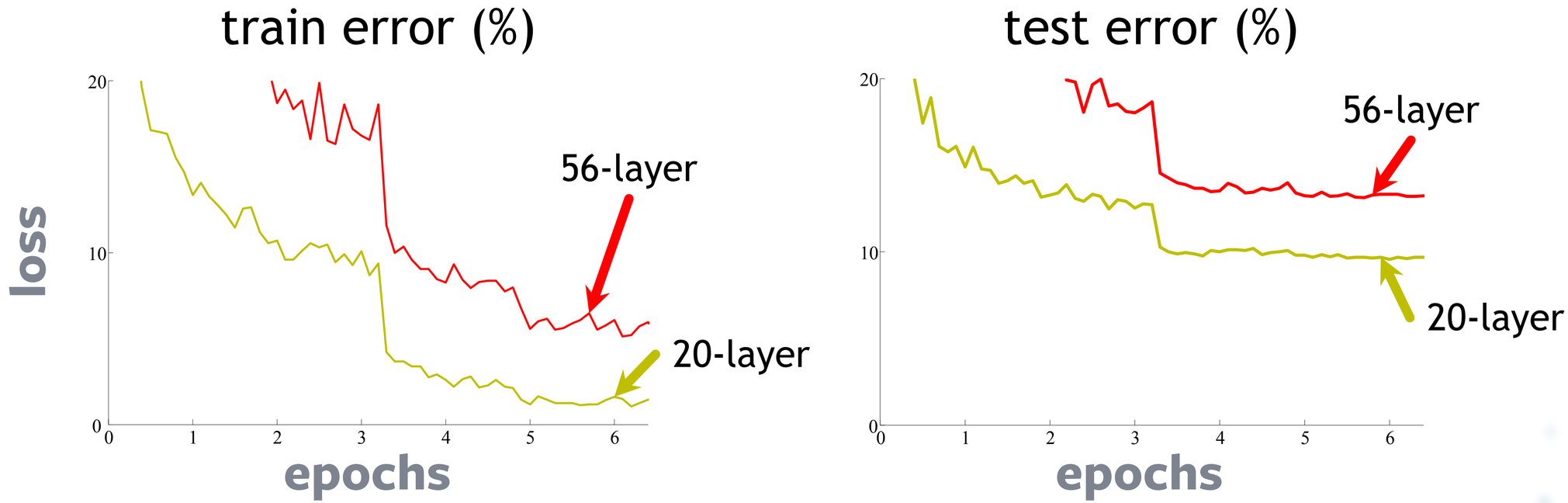
- Simple experiment with stacking many 3x3 convolutional blocks:



* Figure credit: Kaiming He

Stacking layers

- Simple experiment with stacking many 3x3 convolutional blocks:



- Does this suggest overfitting?

* Figure credit: Kaiming He

Stacking layers

- Deeper models should not have higher training error

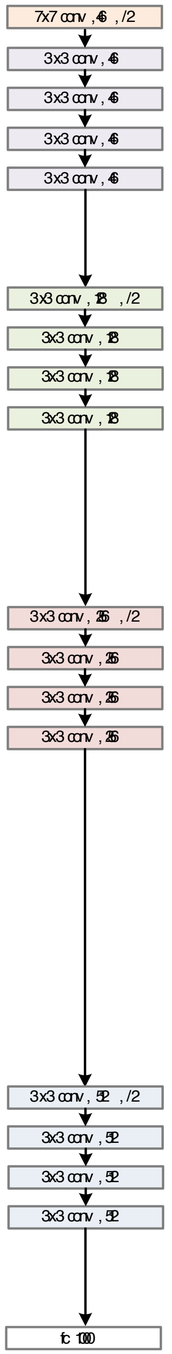
* Figure credit: Kaiming He

Stacking layers

- Deeper models should not have higher training error

- Solution by construction:
  - Train shallow model

"shallow" model

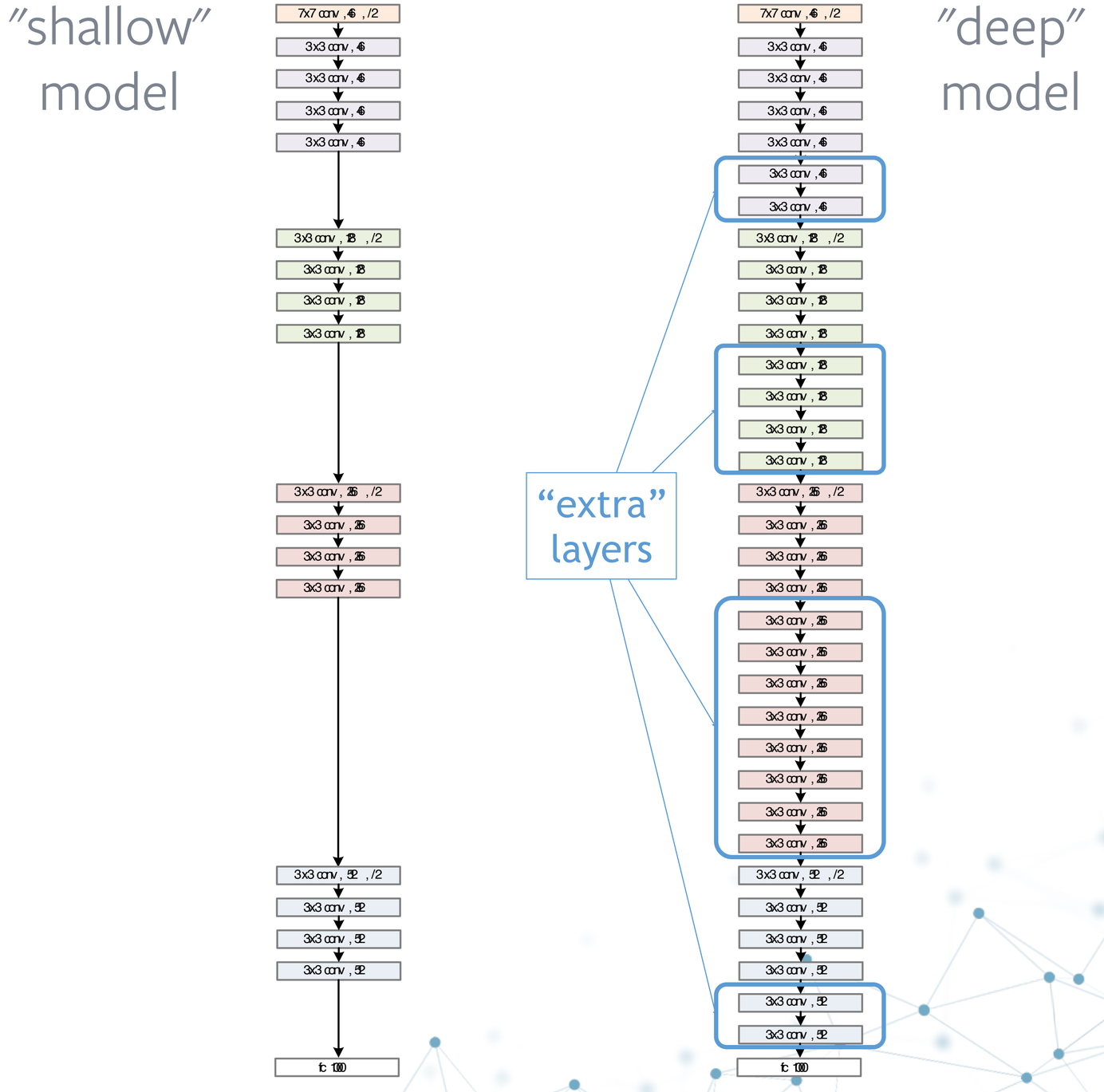


* Figure credit: Kaiming He

Stacking layers

- Deeper models should not have higher training error

- Solution by construction:
  - Train shallow model
  - Add additional layers set to identity
  - Train the deeper model further



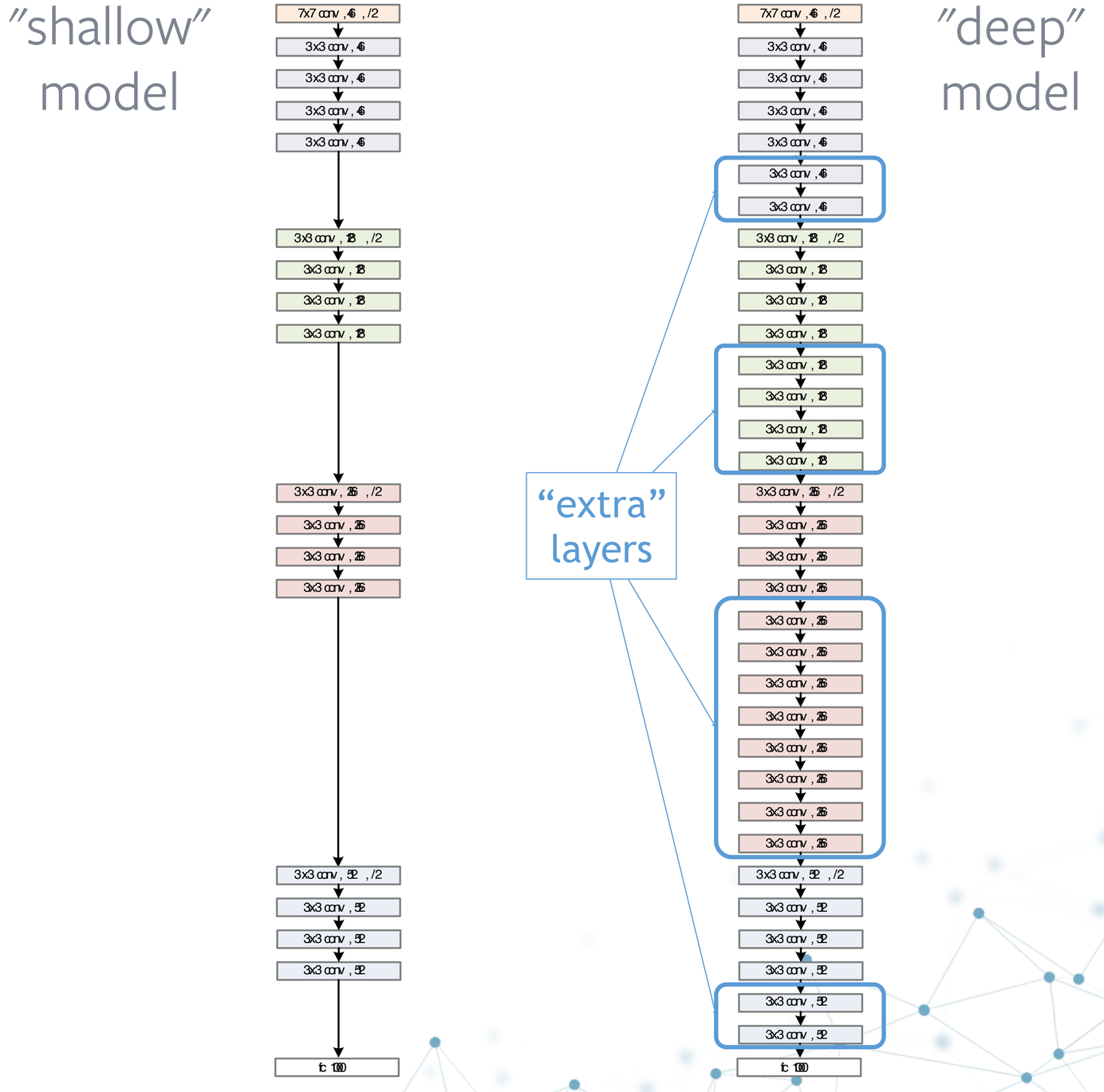
* Figure credit: Kaiming He

Stacking layers

- Deeper models should not have higher training error

- Solution by construction:
  - Train shallow model
  - Add additional layers set to identity
  - Train the deeper model further

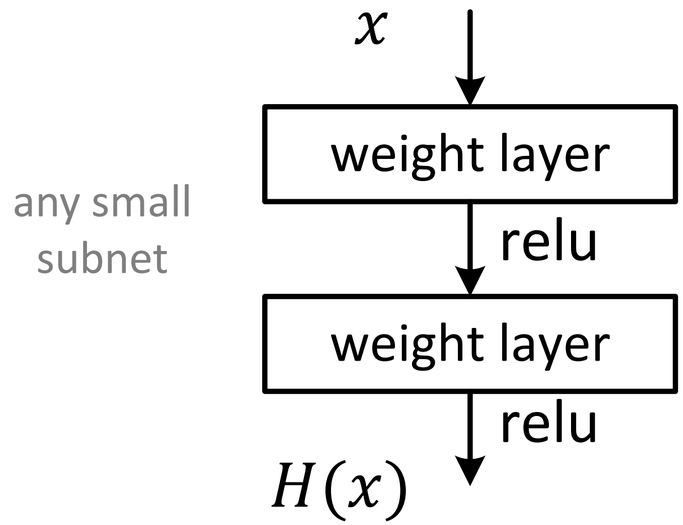
- Learning does not work well :(



* Figure credit: Kaiming He

Residual connections

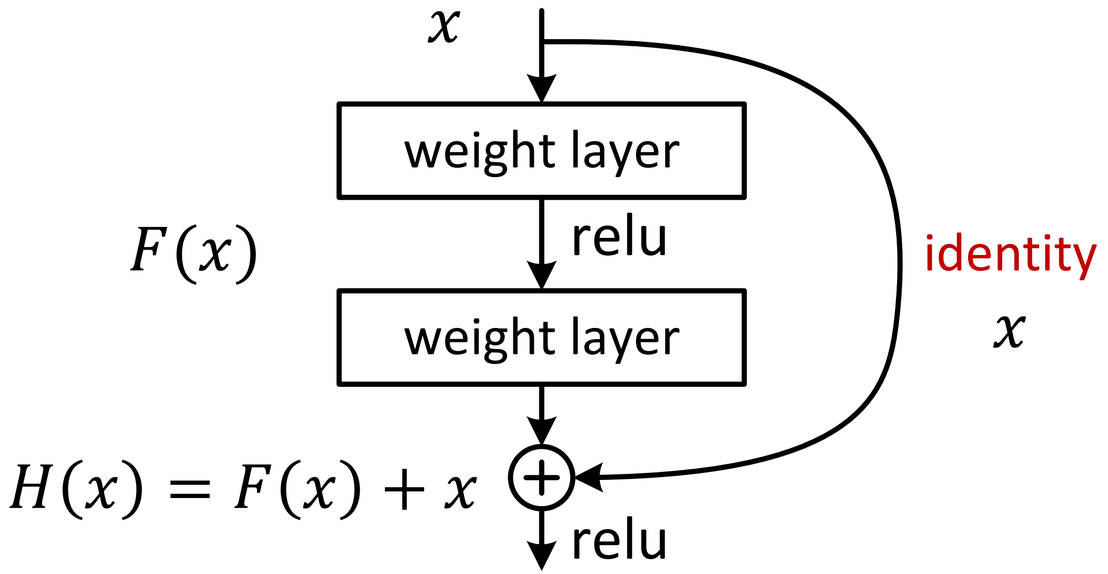
- Block architecture that achieves similar goal
- Take any "regular" network block...



* Figure credit: Kaiming He

Residual connections

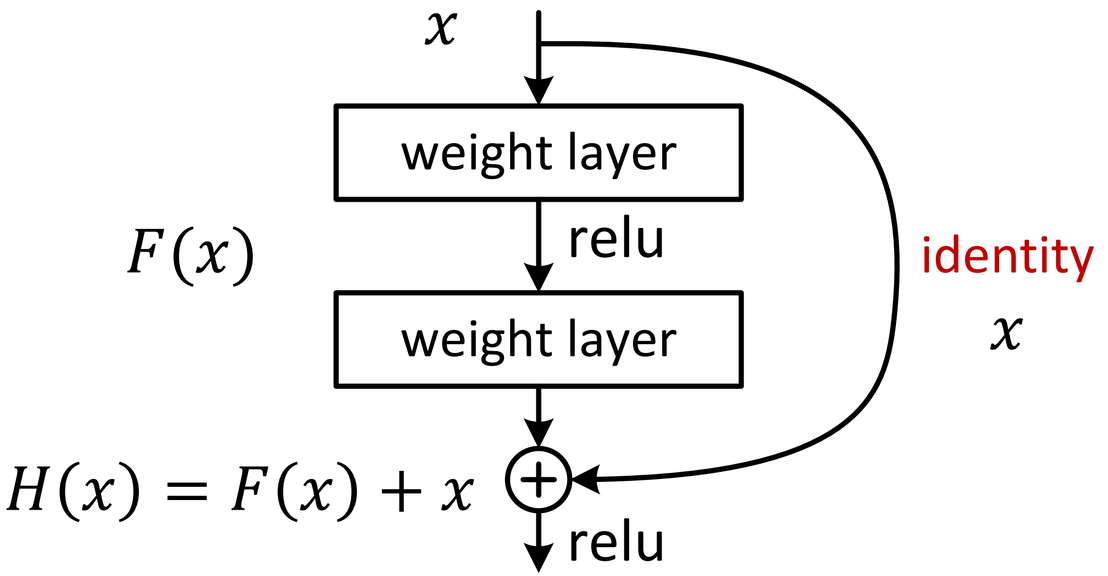
- Block architecture that achieves similar goal
- Take any "regular" network block... and make it **residual**:



* Figure credit: Kaiming He

Residual connections

- Block architecture that achieves similar goal
- Take any "regular" network block... and make it **residual**:

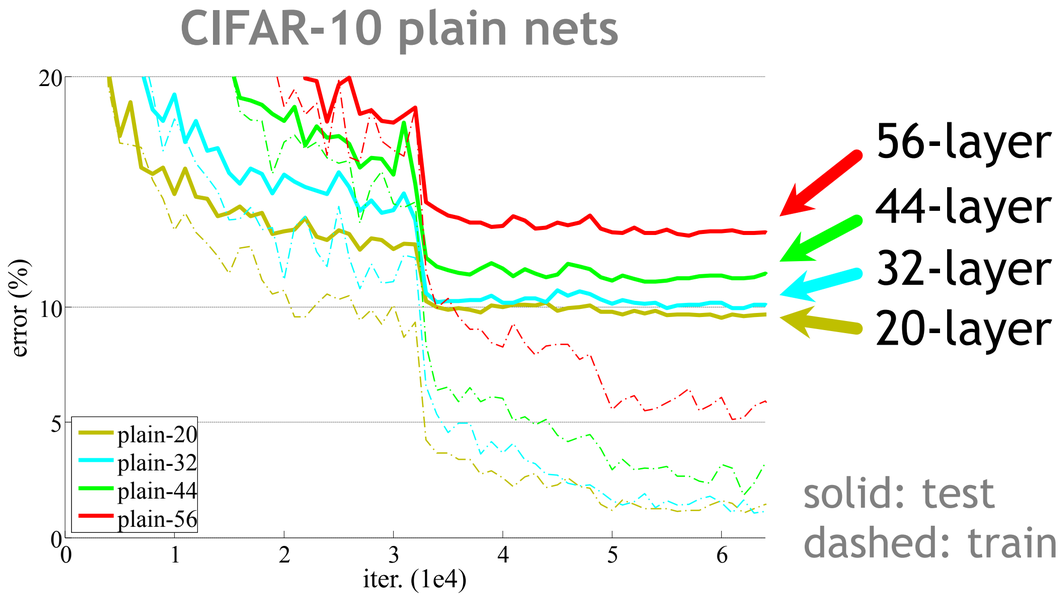


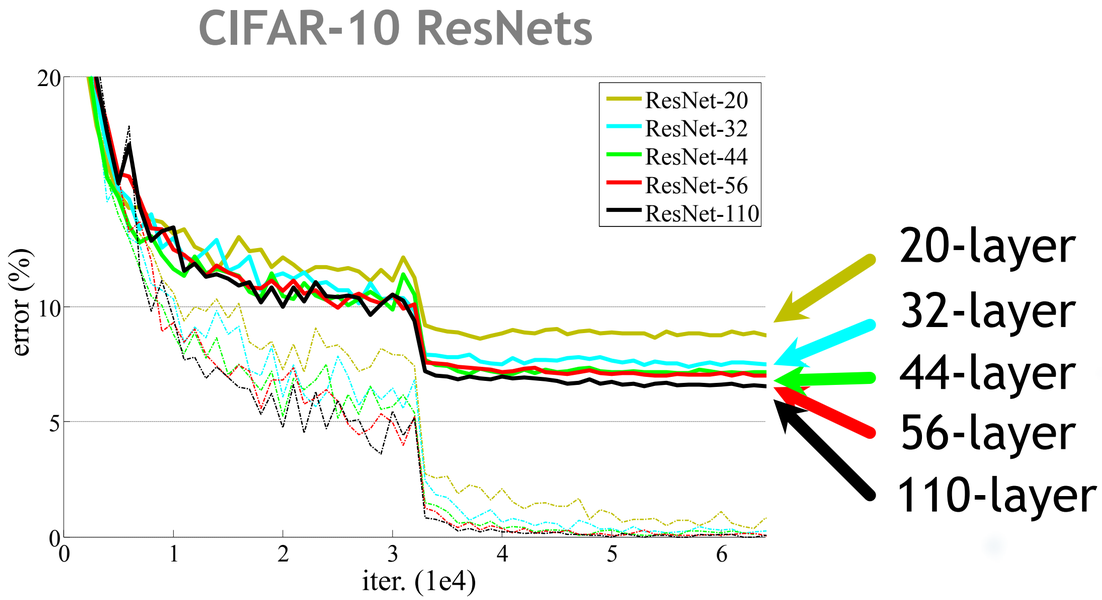
- Initializing weights to zero achieves the desired effect

* Figure credit: Kaiming He

Residual connections

- Deeper models can now be trained without problems:



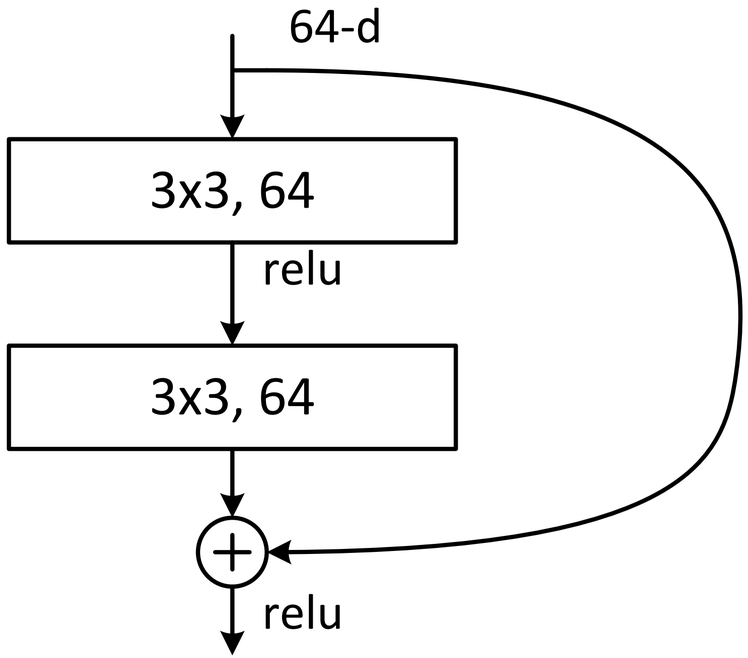


- Overfitting may still happen, but at least training error does not go up

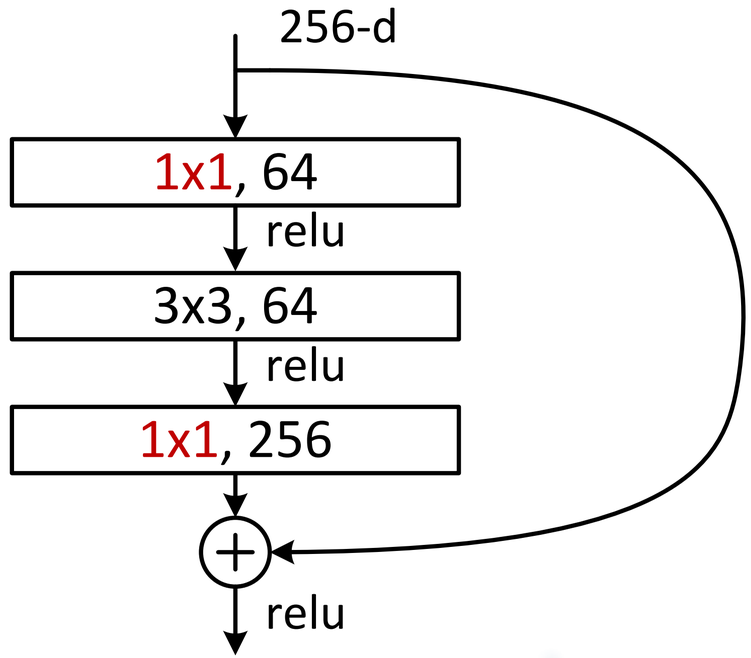
* Figure credit: Kaiming He

Residual networks

- In practice, residual networks (ResNet) use the following block design:



simple design

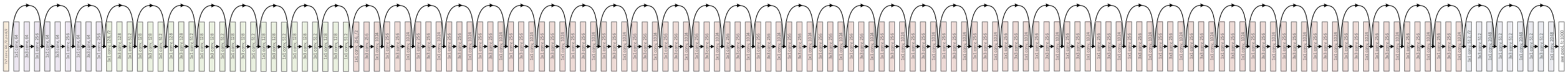


bottleneck design

* Figure credit: Kaiming He

Residual networks

- Full model has pooling layers for downsampling
- All pooling layers do max-pooling but the last one does average-pooling



- Every time the image size halves, the number of channels is doubled

Summary

- Convolutional networks vary in their architecture
- Pooling gradually eliminates spatial structure from the inputs
- Rectified linear units facilitate learning by having good gradients
- Batch normalization controls the scale of the gradients
- Residual helps loss propagation through network, allowing for successfully training deeper networks

Reading material

- S. Ioffe and C. Szegedy. **Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift**. In *International Conference on Machine Learning* (*ICML*), 2015.

- K. He, X. Zhang, S. Ren, and J. Sun. **Deep Residual Learning for Image Recognition**. In *IEEE Conference on Computer Vision and Pattern Recognition* (*CVPR*), 2016.

Questions?